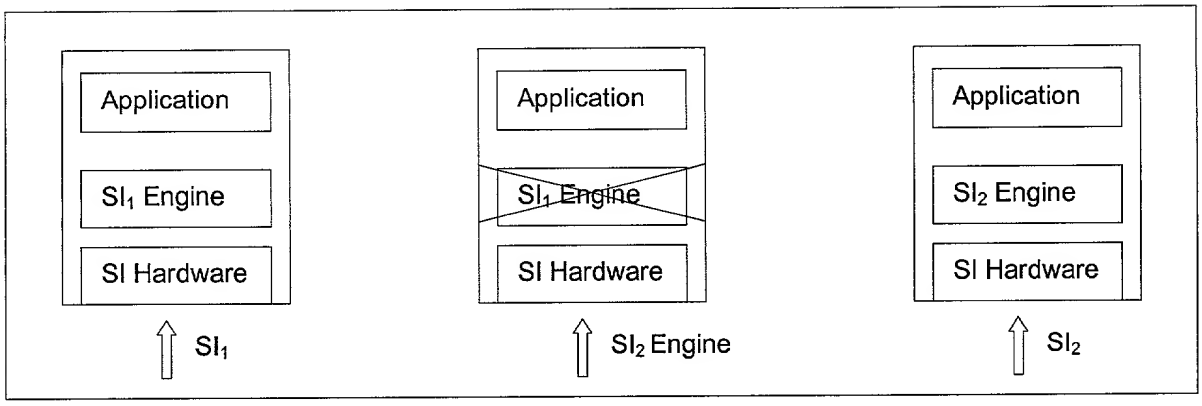
Application
SI1 Engine
SI Hardware
SI1
Application
SI1 Engine
SI Hardware
SI2 Engine
Application
SI2 Engine
SI Hardware
SI2

FIG. 1

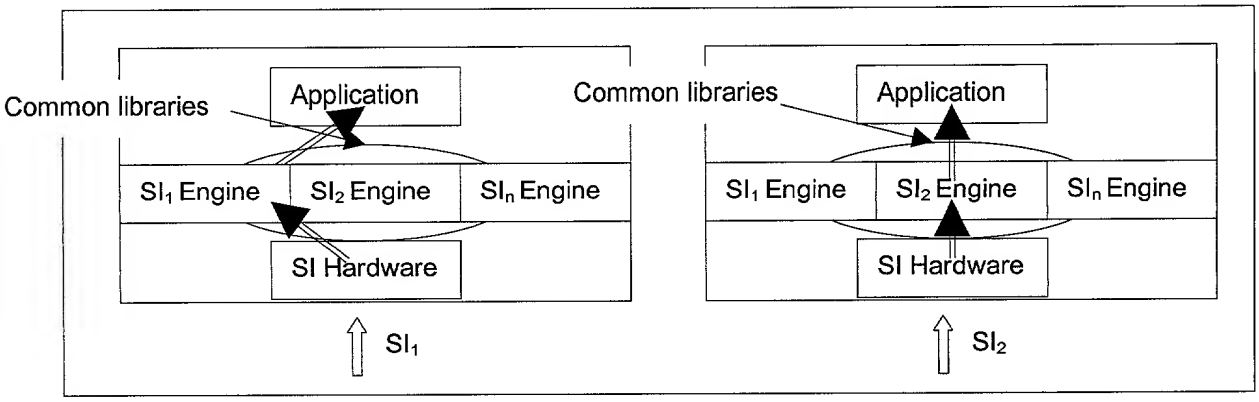
Common libraries
Application
SI1 Engine
SI2 Engine
SIn Engine
SI Hardware
SI1
Common libraries
Application
SI1 Engine
SI2 Engine
SIn Engine
SI Hardware
SI2

FIG. 2

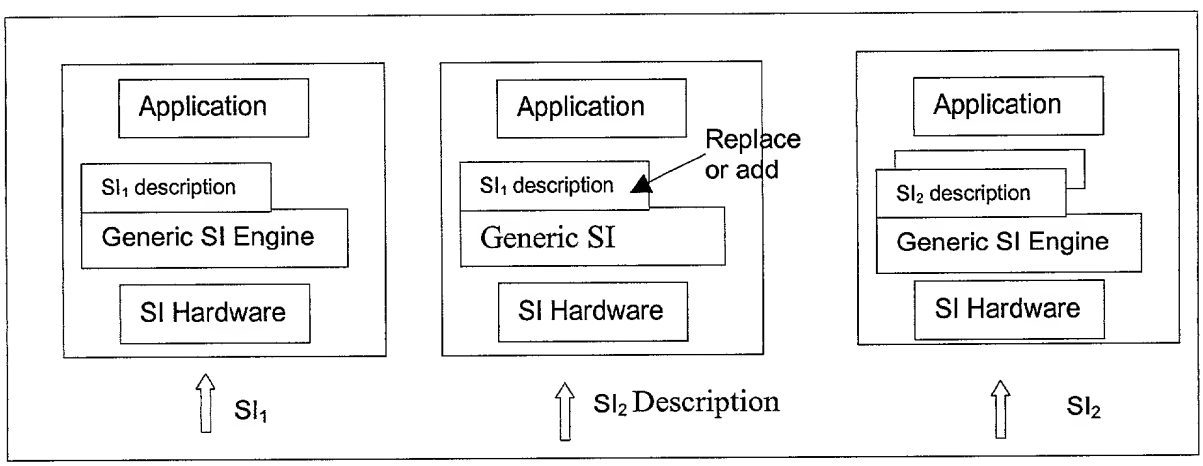
Application
SI1 description
Generic SI Engine
SI Hardware
SI1
Application
Replace
or add
SI1 description
Generic SI
SI Hardware
SI2 Description
Application
SI2 description
Generic SI Engine
SI Hardware
SI2

FIG. 3

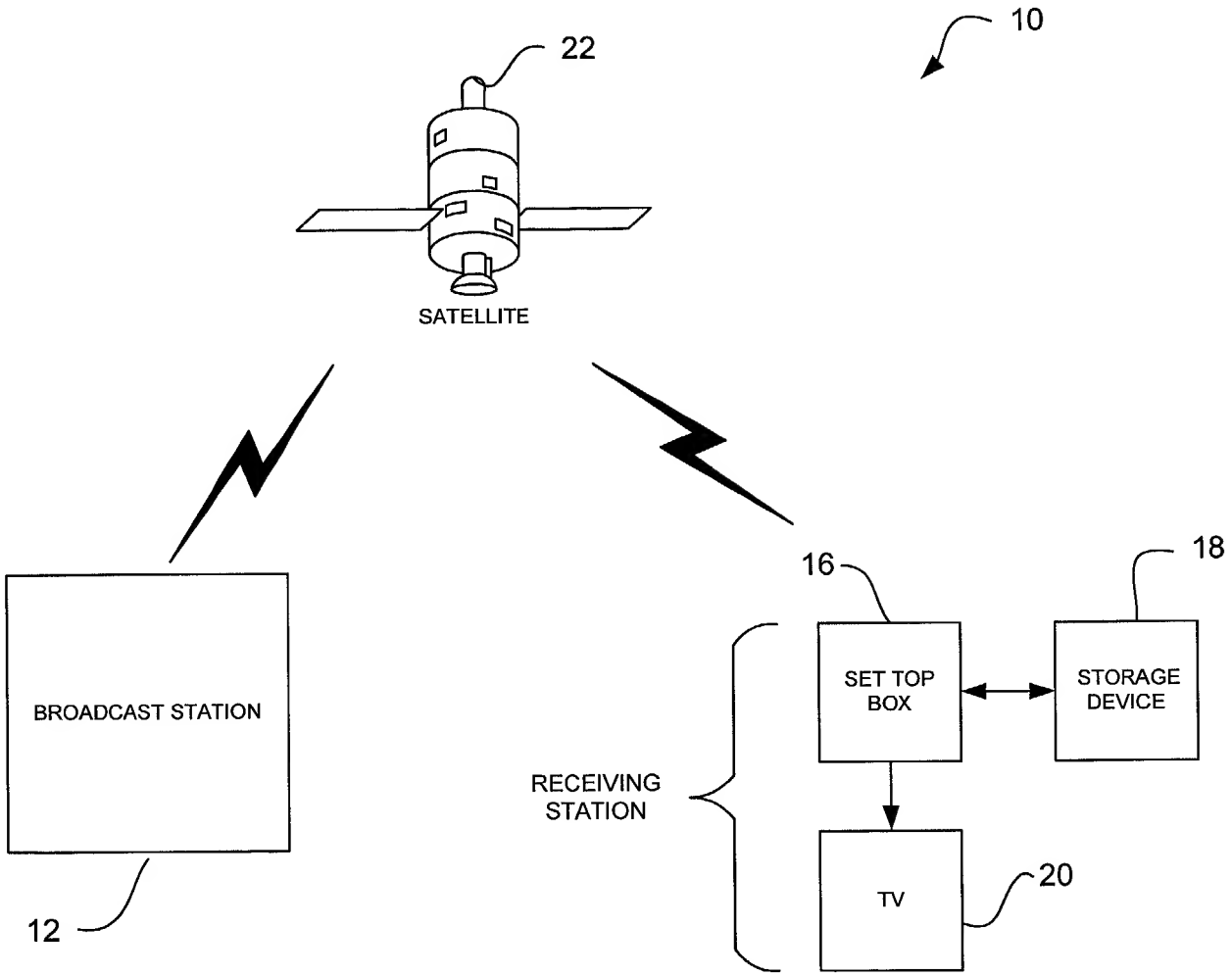
10
22
SATELLITE
BROADCAST STATION
12
16
18
SET TOP BOX
STORAGE DEVICE
RECEIVING STATION
TV
20

FIG. 4

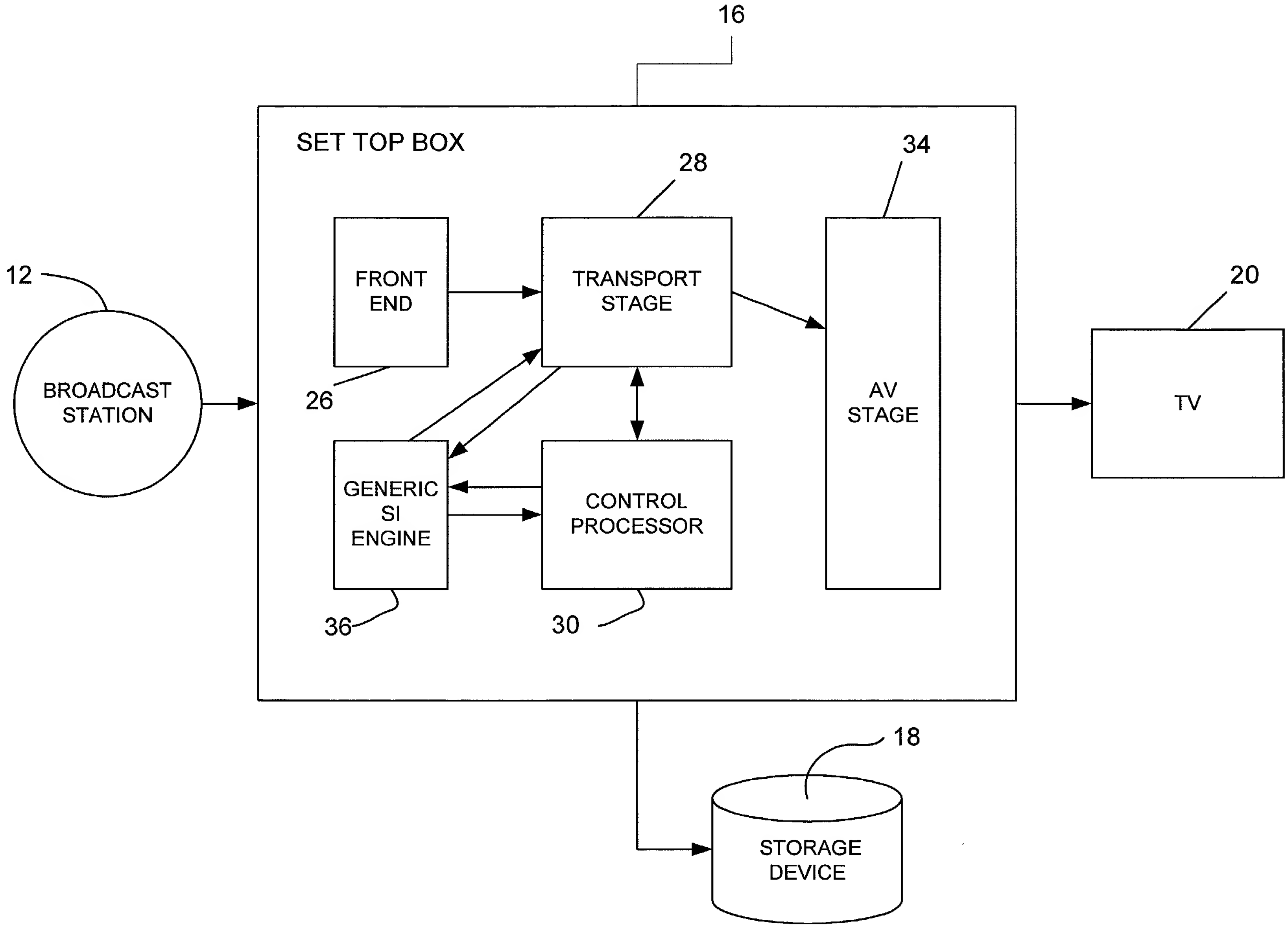
16
SET TOP BOX
28
34
12
FRONT
END
TRANSPORT
STAGE
20
BROADCAST
STATION
26
AV
STAGE
TV
GENERIC
SI
ENGINE
CONTROL
PROCESSOR
36
30
18
STORAGE
DEVICE

FIG. 5

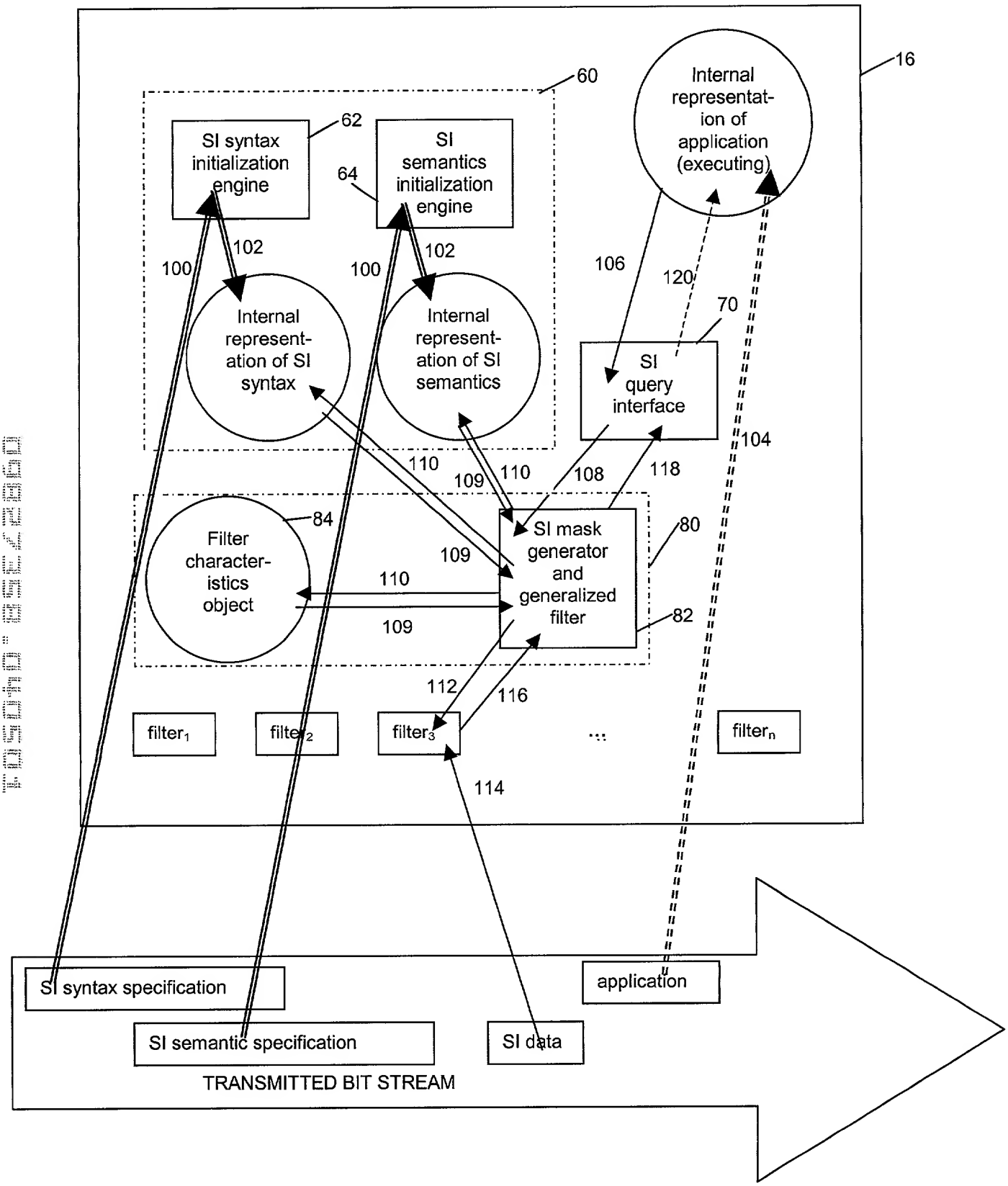
16
60
62
64
SI syntax initialization engine
SI semantics initialization engine
Internal representat-ion of application (executing)
100
102
100
102
106
120
70
Internal represent-ation of SI syntax
Internal represent-ation of SI semantics
SI query interface
104
110
109
110
108
118
84
80
Filter character-istics object
109
SI mask generator and generalized filter
110
109
82
112
116
filter1
filter2
filter3
...
filtern
114
SI syntax specification
application
SI semantic specification
SI data
TRANSMITTED BIT STREAM

FIG. 6

```
Description ::= StructureDefinitions
StructureDefinitions ::= StructureDefinition MoreStructureDefinitions
MoreStructureDefinitions ::= StructureDefiniton | λ
StructureDefinition ::= StructureName "{" StructureBody "}"
StructureName ::= Variable
StructureBody ::= OptHeader FieldDefs
OptHeader ::= "type" "=" TypeName OptFilterValues";" | λ
TypeName ::= Variable
OptFilterValues ::= "," "filterValues" "=" FilterPairs | λ
FilterPairs ::= FilterPair OptMoreFilterPairs
OptMoreFilterPairs ::= FilterPair OptMoreFilterPairs | λ
FilterPair ::= "(" FilterField "," FilterValue ")"
FilterField ::= Variable
FilterValue:: PosInteger
FieldDefs ::= FieldDef OptMoreFieldDefs
OptMoreFieldDefs ::= FieldDef OptMoreFieldDefs | λ
FieldDef ::= SimpleFieldDef  | ComplexFieldDef
SimpleFieldDef ::= FieldName "," FieldSize "," SimpleFieldFormat OptValues ";"
FieldName ::= Variable | "reserved"
FieldSize ::= PosInteger
SimpleFieldFormat ::= PredefinedType
PredefinedType ::= Variable
OptValues ::= "," "=" Value OptValues | λ
Value ::= PosInteger
ComplexFieldDef ::= LoopDef | StructDef SEMICOLON | AlternateDef
LoopDef ::= "loop" "," LoopLength "," "{" FieldDefs "}"
LoopLength ::= Term OptMoreLoopLength
Term ::= PosInteger | StructVariable
OptMoreLoopLength ::= Operation Term OptMoreLoopLength | λ
Operation ::= "+" | "-"
StructVariable ::= Variable OptMoreVariables
OptMoreVariables ::= "." Variable OptMoreVariables | λ
StructDef ::= StructureName "(" ")"
AlternateDef ::= "alternate" "{" FieldDefs "}"
Variable ::= Letter RestrictedCharSet
RestrictedCharSet ::= Letter | Digit | "_" | λ
PosInteger ::= Digit MoreDigits
MoreDigits ::= Digit MoreDigits
Letter ::= "a" | "b" | … | "z" | "A" | "B" | … | "Z"
Digit ::= "0" | "1" | … | "9"
```

FIG. 7

```
network_information_section {
      type = MPEG, filterValues = (PID, 8);
      table_id, 8, uimsbf, = 64, = 65;
      section_syntax_indicator, 1, bslbf;
      reserved, 1, bslbf;
      reserved, 2, bslbf;
      section_length, 12, uimsbf;
      network_id, 16, uimsbf;
      reserved, 2, bslbf;
      version_number, 5, uimsbf;
      current_next_indicator, 1, bslbf;
      section_number, 8, uimsbf;
      last_section_number, 8, uimsbf;
      reserved, 4, bslbf;
      network_descriptors_length, 12, uimsbf;
      loop, network_descriptor_length, {
            NITDescriptor();
      }
      reserved, 4, bslbf;
      transport_stream_loop_length, 12, uimsbf;
      loop, transport_stream_loop_length, {
            transport_stream_id, 16, uimsbf;
            original_network_id, 16, uimsbf;
            reserved, 4, bslbf;
            transport_descriptors_length, 12, uimsbf;
            loop, transport_descriptors_length, {
                  NITDescriptor();
            }
      }
}

NITDescriptor{
      alternate {
            network_name_descriptor();
            service_list_descriptor();
            …
            data_broadcast_id_descriptor();
      }
}
```

FIG. 8

```
struct complexType {
        { this struct would be used to represent any structure or complex field}
        string Name;  {name of the structure or field; used to match value from semantic analyzer}
        complexTypeKind kind; {enumerated type: one of structure, loop, or alternate}

        string baseProtocolType;  { for structures that have types }
                { above field is a handle allowing determination of the name of the lower level filter
                        and the mask representing the bits that the lower level filter is capable of
                        filtering on.}
        couplets *baseProtocolFieldValues;
                { this field shall contain a ptr to a list of memory locations that will contain the filter
                        pairs}
        alternativeList_t *alternatives;
                { pointer to a list of all of the possible alternatives for fields of this field or structure }

}tableType_t;

struct alternativeList {
        string Name;
        fieldDescription_t *fieldDescriptions
        { this would be a pointer to a linked list of lists of descriptions of the alternate lists of fields;
        since it is possible to know how many elements need to be in this list, space for it can be
        dynamically allocated and efficient access using pointer arithmetic is possible. }
        alternativeList_t *next;
}alternativeList_t;

struct fieldDescription {
        { this particular representation assumes that exactly enough space to hold all of the fields is
        allocated allowing direct access to any field without walking through a linked list, simply by
        using pointer arithmetic  -- hence, there is no "next" field at the end.}
        string fieldName;   { this is also used to match value from semantic analyzer.}
        fieldTypeKind_t fieldTypeKind;  { enumerated type: one of basic, structure, or loop.}
        fieldType_t *fieldType; { if basic, this is a pointer to one of the basic types listed in the
                system encoding type tables;  in example above, these would be uimsbf and bslbf.
                if structure or loop, this field is a pointer to a linked list of tableType_t structures, any of
                which could occupy this field.}
        indexList_t *lengthList { see below for definition of structure pointed to by this field; allows the
                length of the field to be a fixed constant, a value provided in another field (of any
                structure or loop), or an arbitrary sum or difference of any of these. }
        indexList_t *offsetList {same format as lengthList, and computed/computable from the
                lengthLists, but storing this valueallows direct access to this fieldDescription}
        value_list_t *valueList; { this is a list of values that have been specified for this field}
}fieldDescription_t;

struct indexList{
        enum valueKind; { one of fixed_constant, reference_field or variable.
        fixed_constant means a fixed number of bits;  reference_field means that the length is given
        by another field in this (or another) structure, and variable means that it is not known.}
        valuePtr_t *valuePtr;{ this would be a pointer to an integer if type is fixed_constant, another
                field if type is reference_field; or pointer is null if type is variable}

        int multiplier; { either +1 or –1; allows arbitrary sums/differences for length }
        indexList *next;   { in case this value is a sum of other values}
}indexList_t;
```

FIG. 9

```
Define int Movie = 1;
Define string Actor = 'Actor';
Define string ProductionCompany = 'Production Company';

Define Object eventInfo = {
fetch(event_information_section,
filter:
table_id > 90,
table_id <= 113;
return:
name := loop_1 = outer_loop.loop1.extended_event_descriptor
= named_descriptor.loop_2.text_char[],
start_time := outer_loop.start_time,
end_time := compute(outer_loop.start_time + outer_loop.duration)
)

where

channel(relop relationalOp, int value) =
{obtain triplet := identifyingTrio where channel(relationalOp, value);
set(event_information_section,
      filter:
      original_network_id == triplet.original_network_id,
      transport_stream_id == triplet.transport_stream_id,
      service_id == triplet.service_id);
}
/* See below for the definition of the identifyingTrio object */

eventType(relop relationalOp, int value) =
{set(event_information_section,
      filter: outer_loop.loop1.descriptor.content_descriptor.loop_1.
      content_nibble_1 relationalOp value); }

instantiate(relop relationalOp1, string str1, relop relationalOp2,
      string str2) =
{set(event_information_section,
      filter:
      named_descriptor.loop_1=chosen_loop.item_description_char
      relationalOp str1,
      chosen_loop1.item_char relationalOp2 str2); }

startTime(relop relationalOp, int t1) =
/* assumes that application program has already converted the
requested t1 into seconds past midnight local time.
Also, assumes that the following global variables are known,
all in seconds */

{set (event_information_section, filter: section_number >=
compute( ( (t1-Current_local_time) +
( ( Current_local_time - UTCDifference) mod 24) div 3) * 8),
outer_loop.start_time relationalOp t1);}

endTime(relop relationalOp, int t2) =
{set(event_information_section,
filter:
section_number <= compute ( ( ( ( ( t2 - Current_local_time) +
      ( ( Current_local_time - UTCDifference) mod 24 ) )
```

FIG. 10a

```
      div 3) * 8) + 7),
      compute(outer_loop.start_time + outer_loop.duration)
      relationalOp t2);
}

event_id(relop relationalOp, int value) =
{set(event_information_section,
      filter: outer_loop.event_id relationalOp value); }
}

Define Object identifyingTrio = {
fetch(channel_correspondence_section,
return:
original_network_id :=
      loop_1 = channel_loop.original_network_id,
transport_stream_id := channel_loop.transport_stream_id,
service_id := channel_loop.service_id)
where
channel(relop relopetionOp, int value) =
      {set(channel_correspondence_section,
      filter: channel_number relationalOp value);
}
}
```

FIG. 10b

```
Program ::= Definitions
Definitions ::= Definition OptMoreDefinitions
OptMoreDefinitions::= λ | Definition OptMoreDefinitions
Definition ::= DEFINE RestOfDef
RestOfDef ::= ConstantDef  | ObjectDef
ConstantDef ::= Type VARIABLE EQUALS Value SEMICOLON
Type ::= INTTYPE | STRINGTYPE
Value ::= INTEGER| STRING
ObjectDef ::= OBJECT ObjName EQUALS LEFTCURLY Fetches OptRestObjDef  RIGHTCURLY
ObjName ::= VARIABLE
Fetches ::= Fetch OptMoreFetches
OptMoreFetches ::= λ | Fetch OptMoreFetches
Fetch ::= FETCH LPAREN StructureName COMMA OptFilterPart ReturnPart RPAREN
StructureName ::= VARIABLE
OptFilterPart::= λ | FilterPart
ReturnPart ::= RETURN COLON ReturnStmts
ReturnStmts ::= ReturnStmt OptMoreReturnStmts
OptMoreReturnStmts ::= λ | COMMA ReturnStmt OptMoreReturnStmts
ReturnStmt ::= FieldName ASSIGNOP FieldValue
FieldName ::= VARIABLE
FieldValue ::= PathValue | ComputeExpression
PathValue ::= PathName OptArrayInd
OptArrayInd ::= λ | LSQUARE RSQUARE
PathName ::= NarrowName OptTempName OptRestOfPath
OptTempName ::= λ | EQUALS TempName
NarrowName ::= VARIABLE
TempName ::= VARIABLE
OptRestOfPath ::= λ | DOT PathName
ComputeExpression ::= COMPUTE LPAREN Expr RPAREN
Expr ::= Term RestOfExpr
RestOfExpr ::= λ | BinOperator Expr
Term::= CompoundTerm | SimpleTerm
CompoundTerm ::= LPAREN Expr RPAREN
SimpleTerm ::= INTEGER | PathName
BinOperator ::= PLUS | MINUS | TIMES | DIV | MOD | MIN | MAX
OptRestObjDef ::= λ | WHERE Methods
Methods ::=        Method OptMoreMethods
OptMoreMethods ::= λ | Method OptMoreMethods
Method ::= MethodName LPAREN ParamList RPAREN EQUALS MethodDefn
MethodName ::= VARIABLE
ParamList ::= λ  | Parameter OptMoreParameters
OptMoreParameters ::= λ  | COMMA Parameter OptMoreParameters
Parameter ::= SpecifiedArg | ConstantSpec | RelSpec
RelSpec ::= RELOP RelOpVariable
SpecifiedArg ::= VARIABLE
ConstantSpec ::= Type ConstantVar
ConstantVar ::= VARIABLE
RelOpVariable ::= VARIABLE
MethodDefn ::= LEFTCURLY OptAcquisitions FilterSets RIGHTCURLY
OptAcquisitions ::= λ |  Acquisition OptAcquisitions
FilterSets ::= FilterSet OptMoreFilterSets
OptMoreFilterSets ::= λ | FilterSet OptMoreFilterSets
FilterSet ::= SET LPAREN StructureName COMMA FILTER COLON Filters RPAREN SEMICOLON
FilterPart ::= FILTER COLON Filters SEMICOLON
Filters ::= CompOperand OptArrayInd Comparator CompOperand  OptMoreFilters
```

FIG. 11a

OptMoreFilters ::= λ | COMMA Filters
Comparator ::= RelOpVariable | DBLEQUALS | GT | GTEQ | LTEQ | LT | NOTEQUALS
CompOperand ::= PathName | ComputeExpression | INTEGER | STRING
Acquisition ::= OBTAIN ObjectInstantiation ASSIGNOP ObjName OptConstraint SEMICOLON
OptConstraint ::= WHERE OptConstraints
ObjectInstantiation ::= VARIABLE OptArrayInd
OptConstraints ::= λ | Constraint OptConstraints
Constraint ::= MethodName LPAREN ActualParamList RPAREN
ActualParamList ::= ActualParam OptMoreActualParams
OptMoreActualParams ::= λ | COMMA ActualParam OptMoreActualParams
ActualParam ::= VARIABLE | INTEGER

FIG. 11b

```
DefinionList_t *PtrToDefns;

struct DefinitionList {
      Defn_t *DefPtr;
      DefinitionList_t *NextDefn;
}DefinitionList_t;

struct Defn{
      enum DefnType; {One of "Integer", "String", or "Object"}
      string Name; {VARIABLE from ConstantDef or ObjName from
            ObjectDef}
      void *valuePtr; {points to an integer, a string or  ObjParts_t
            structure (defined below)}
} Defn_t;

struct ObjectParts{
      FetchList_t *FList;
      MethodList_t *MethodList;
}ObjParts_t;

struct MethodList{
      MethodStruct_t *MethodPtr;
      MethodList_t *NextMethod;
} MethodList_t;


struct FetchList{
      FetchStruct_t *FStruct;
      FetchList_t *NextFetch;
}FetchList_t;

struct FetchStruct{
      string StructureName;
      FilterList_t *FiltList;
      ReturnList_t *RtrnList;
}FetchStruct_t;

struct FilterList{
      FilterStruct_t *FiltStruct;
      FilterList_t *NextFilter;
}FilterList_t;

struct FilterStruct{
      string StructureName;
      FilterFieldList_t *FilterFields;
}FilterStruct_t;

struct FilterFieldList{
      FilterFieldStruct_t *FilterField;
      FilterFieldList_t *NextFilterField;
}FilterFieldList_t;

struct FilterFieldStruct{
      void *FirstOperand; { points to one of a pathList_t,
            ComputeStack_t, an integer, or a string }
      void *RelationalOp; { points to a string (if operator can
```

FIG. 12a

```
            vary) or an enum containing one of eq, gt, gteq, etc. )}
      enum RelationalType; { one of "string" or "actual"}
      void *SecondOperand; { same type as FirstOperand }
}FilterFieldStruct_t;

struct ReturnList{
      ReturnStruct_t *RtrnStruct;
      ReturnList_t *NextRtrn;
}ReturnList_t;

struct ReturnStruct{
      string NewFieldName;
      void *FieldValue; { ptr to one of a pathList_t or ComputeStack_t}
      enum type; { one of "path" or "computeStack" }
}ReturnStruct_t;

struct pathList{
      string Name;
      string tempName; {used in scope to mean that a single
            (sub-)structure is multiply constrained}
      pathList_t *nextName;
      enum Type; { one of "array" or "scalar"}
} pathList_t;

struct ComputeStack{
      void *ItemPtr; {points to item to push on list}
            {Item is of type integer, pathname, or operator}
            {operator is an enum type containing one of:
                  plus, minus, times, div, min, max, or mod}
      enum type; { one of "integer", "pathname" or "operator" }
      ComputeStack_t *nextToPushOn;
}ComputeStack_t;

struct MethodStruct{
      string MethodName;
      ParamList_t *ParamListPtr;
      MethodDefnStruct_t *MethodDefn;
}MethodStruct_t;

struct MethodDefnStruct{
      AcqList_t *AcqList;
      FilterList_t *FilterList;
}MethodDefnStruct_t;

struct AcqList{
      AcqStruct_t *AcqPtr;
      AcqList_t *NextAcq;
}AcqList_t;

struct AcqStruct{
      string NewObjName;
      enum ObjType; { one of "scalar" or "array"}
      string OldObjName;
      ConstraintList_t *Constraints;
} AcqStruct_t;

struct ConstraintList{
```

FIG. 12b

```
      ConstraintStruct_t *Constraint;
      ConstraintList_t *nextConstraint;
}ConstraintList_t;

struct ConstraintStruct{
      string MethodName;
      ActualParamList_t *ActualParams;
} ConstraintStruct_t;

struct ActualParamList{
      void *ActualParam; { points to a string or an int}
      enum type; {one of "string" or "integer" }
      ActualParamList_t *NextParam;
}ActualParamList_t;

struct ParamList{
      void_t *Param; {ptr to type SpecifiedArg_t, ConstantSpec_t, or
            RelSpec_t}
      enum type; { one of SpecifiedArg, ConstantSpec, or RelSpec}
      ParamList_t *NextParam;
}ParamList_t;

struct SpecifiedArg{
      string Argument;
}SpecifiedArg_t;

struct ConstantSpec{
      enum type; { one of "integer" or "string"}
      void *value; { pointer to an integer or a string}
} ConstantSpec_t;

struct RelSpec{
      string RelOpVariable;
}RelSpec_t;
```

FIG. 12c

```
Select all SoughtLoop = loop_1 from event_information_section where
        // event is of type "news"
        (content_nibble_1 from content_descriptor from  SoughtLoop = = newstype)
                //where newstype is a pre-defined constant that has been assigned the value 2
        and
        // and it is in the same bouquet as the current actual_transport_stream
        (transport_stream_id = =
                (Select transport_stream_id from MidLoop = loop_2 from
                CurrentBouquet = bouquet_association_section where
                        ((Select transport_stream_id
                                from OtherMidLoop = loop_2  //ok =MidLoop
                                from CurrentBouquet = =
                                        (Select transport_stream_id from
                                        Actual_NIT = network_information_section
                                        where table_id = = 64)) and

                        (Select original_network_id
                                from OtherMidLoop
                                from CurrentBouquet = =
                                (Select network_id from Actual_NIT)))

        and
        (original_network_id = =
                (Select original_network_id from MidLoop from CurrentBouquet))
        // and it is carried on cable (as opposed to satellite or terrestrial)
        and
        transport_stream_id = =
                (Select transport_stream_id from DeliveryLoop = loop_2 from
                        AnyNIT = network_information_section where
                        exists cable_delivery_system_descriptor from DeliveryLoop from AnyNIT)
        and
        original_network_id ==
                (Select original_network_id from DeliveryLoop from AnyNIT)
        // and its time is within the requested range
        and
        DVB_time_Between(RequestedStartTime, RequestedEndTime, SoughtLoop.start_time,
                SoughtLoop.duration)
```

FIG. 13

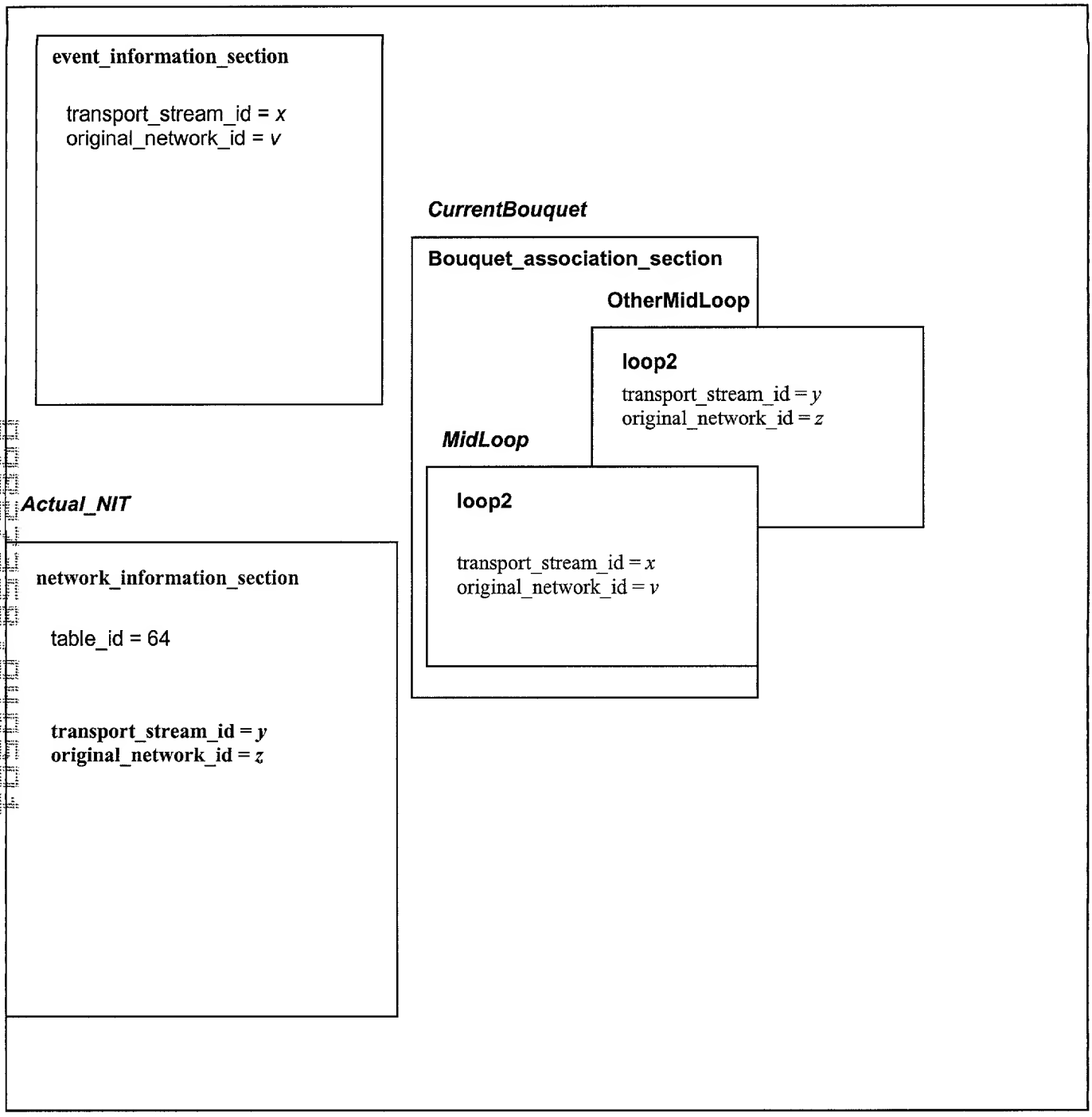
event_information_section
transport_stream_id = x
original_network_id = v
CurrentBouquet
Bouquet_association_section
OtherMidLoop
loop2
transport_stream_id = y
original_network_id = z
MidLoop
loop2
transport_stream_id = x
original_network_id = v
Actual_NIT
network_information_section
table_id = 64
transport_stream_id = y
original_network_id = z

FIG. 14

```
        //
        // expansion of DVB_time_Between from previous figure
        // (almost complete – see last comment below for a few details
        // intentionally left out)
        start_time >= RequestedStartTime
        and
        start_time + duration <= RequestedEndTime
        and
        time_since_midnight = CurrentLocalTime – (TimeDiff) (mod 24)
        // CurrentLocalTime and the offset from UTC-0 is generally cached within a
        // set-top box – if they are not they can be obtained from the TOT or TDT tables.
        and
        StartInUTC0 = max(0, (RequestedStartTime – CurrentLocalTime)-
time_since_midnight))
        and
        EndInUTC0 = max(96, (RequestedEndTime – CurrentLocalTime) –
time_since_midnight)
        and
        (StartInUTC0 div 3) * 8 <= section_number <= ((EndInUTC0 div 3) * 8) + 7
        and
        this_section_number = section_number <= last_section_in_segment_number from
event_information_section where
                first_section_in_segment = section_number from
event_information_section where
                        first_section_in_segment <= this_section_number and
                        first_section_in_segment div 8 = = 0
        // Should be code in the beginning converting all of the parameters and
        // known constants (including RequestedStartTime, RequestedEndTime,
        // CurrentLocalTime, and TimeDiff) as well as the time values extracted from
        // each event_information_section loop (including start_time and duration) to
        // the number of minutes since midnight GMT, January 1, 1970 – or some
        //  other reference date.  It is the converted values that are expected to be
        // used for the comparisons above.
```

FIG. 15

```
ActualTransportStreamID(X) :-
        program_association_table(A), is_field(A, transport_stream_id, X).
// X is the actual transport stream id if the program association section (A) has, as a
// transport_stream_id field with a value of X.

TransportStreamInBouquet(C, B) :-
        bouquet_association_section(L),
        is_field (L, bouquet_id, B),
        is_field(L, transport_stream_id, C).
// C is in Bouquet with bouquet_id B if a bouquet_association_section L has a bouquet_id
// field with a value of B and a transport_stream_id field with a value of C.

TransmissionMedia(X, media) :-
        ActualTransportStreamID(X),
        // case 1: X is the actual transport stream
        network_information_section(N),
        is_field(N, table_id, 32),
        is_field(N, loop1, L),
        is_field(L, descriptor, D),
        frequency_list_descriptor(D)
        is_field(D, coding_type, media).

// The transmission media of transport stream X is "media" if X is the actual transport
// stream ID, N is a network_information_section for the current stream, L is loop 1 of N,
// D is a frequency_list_descriptor in L, and D has coding_type field of that "media. "

TransmissionMedia(X, media) :-
        network_information_table(N),
        is_field(N, loop2, L),
        is_field(L, transport_stream_id, X),
        is_field(L, descriptor, D),
        frequency_list_descriptor(D)
        is_field(D, coding_type, media).

// Another possibility for X having a transmission media "media"
// Here N can be any network_information_table where the second loop contains the
// transport_stream_id in question and it has a coding_type field of value that "media."
```

FIG. 16

```
GetAllEITEvents(X, [LocalStartTime,LocalEndTime], DefiniteEvents) :-
        // This rule tells how to obtain all EITs that contain schedules for the transport
        // stream whose id is X for events that occur within the local time range.
SchedulesAreBroadcast(X),
        // step 1: determine whether any of the services carried on the transport stream
        // whose id is X has its schedule broadcast.  If it does not, there
        // will be no such EITs, report this.
ObtainLocalTime(CurrentTime),
LocalEndTime > CurrentTime,
ComputeOffsetRange([LocalStartTime, LocalEndTime], CurrentTime, OffsetRange),
        // step 2: determine the local time and how far ahead of this (at least part of) the
        // range is.  If the entire range is past, report an error. Otherwise, we have a new
        // range, which we'll call an offset range (OffsetStart, OffsetEnd).
GetUTC_0Time(CurrentUTC_0),
        // step 3: Determine the current time in UTC-0, assuming that it is transmitted
        // as a 24 hour clock, so it is the # of hours since midnight "today."
TimeSince12Range(CurrentUTC_0, OffsetRange, RangeSince12),
        // step 4: Add the hours since midnight obtained in step 3 to both the OffsetStart
        // and OffsetEnd to determine the block of time since midnight UTC-0 for which
        // schedules are desired.
TranslateTimeToEITSegmentNumber(RangeSince12, SegmentStart, SegmentEnd),
        // step5: Determine the segment range and request the EIT scheduled in those
        // segment ranges.
GetEITEventsInRange(SegmentStart, SegmentEnd, X, PossibleEvents),
CheckTimes(LocalStartTime, LocalEndTime, PossibleEvents, DefiniteEvents).
```

FIG. 17a

```
SchedulesAreBroadcast(X) :-
        service_description_table(S), is_field(S, transport_stream_id, X),
        is_field(S, loop1, L), is_field(L, EIT_schedule_flag, Flag),
        Flag == 1.
```

FIG. 17b

```
GetEITEventsInRange(SegmentStart, SegmentEnd, X, Events) :-
        SegmentEnd > SegmentStart,
        event_information_section(S), is_field(S, transport_stream_id, X),
        is_field(S, section_number, SegmentStart),
        is_field(S, segment_last_section_number, SLN),
        is_field(S, loop_1, L), Append( L, Events),
        GetRestOfEITSegment(SegmentStart+1, SLN, X, Events),
        NewStart = SegmentStart + 8,
        GetEITsInRange(NewStart, SegmentEnd, X, Events).

GetRestOfEITSegment(SegmentStart, SegmentEnd, X, Events) :-
        SegmentEnd >= SegmentStart,
        event_information_section(S), is_field(S, transport_stream_id, X),
        is_field(S, section_number, SegmentStart),
        is_field(S, segment_last_section_number, SLN),
        is_field(S, loop_1, L), Append( L, Events),
        GetRestOfEITSegment(SegmentStart+1, SLN, X, Events).
```

FIG. 17c

```
ObtainLocalTime(CurrentTime) :-
        time_offset_section(S), is_field(S, UTC_time, CurrentMJD_UTC_0),
        is_field(S, loop1, L), is_field(L, descriptor, D),
        local_time_offset_descriptor(D), is_field(D, local_time_offset_polarity, P),
        is_field(D, local_time_offset, Offset),
        ConvertToHoursSince_1_1_1900(CurrentMJD_UTC_0, CurrentTime, P, Offset).
        // ConvertToHoursSince1900 not shown – can be implemented using arithmetic
        // formulas copied, nearly identically from DVB EN300_468, Annex C – note
        // that  (-1) is raised to the P power and multiplied by the offset.

GetUTC_0Time(CurrentUTC_0) :-
        time_date_section(S), is_field(S, UTC_time, CurrentMJD_UTC_0),
        ConvertToHoursSince_1_1_1900(CurrentMJD_UTC_0, CurrentUTC_0, 0, 0).

ComputeOffsetRange([LocalStartTime, LocalEndTime], CurrentTime, [OffsetStart,
OffsetEnd]) :-
        LocalStartTime > CurrentTime, OffsetStart = LocalStartTime – CurrentTime,
OffsetEnd =
        LocalEndTime – CurrentTime.
        // if entire range is after the current time – otherwise, see next rule

ComputeOffsetRange([LocalStartTime, LocalEndTime], CurrentTime, [OffsetStart,
OffsetEnd]) :-
        OffsetStart = 0, OffsetEnd = LocalEndTime – CurrentTime.

TimeSince12Range(CurrentUTC_0, [OffsetStart, OffsetEnd], [StartTimeSince12,
EndTimeSince12]):-
        StartTimeSince12 = CurrentUTC_0 + OffsetStart,
        EndTimeSince12 = CurrentUTC_0 + OffsetEnd.
```

FIG. 17d

```
TranslateTimeToEITSegmentNumber([StartTimeSince12, EndTimeSince12],
SegmentStart, SegmentEnd):-
        SegmentEnd = ( (EndTimeSince12 div 3) * 8 ) +7,
        SegmentEnd < 256,
        SegmentStart = (StartTimeSince12 div 3) * 8).
// use above if all times in range – otherwise use below.

TranslateTimeToEITSegmentNumber([StartTimeSince12, EndTimeSince12],
SegmentStart, SegmentEnd):-
        SegmentEnd = ( (EndTimeSince12 div 3) * 8 ) +7,
        SegmentEnd > 255,  SegmentEnd = 255,
        SegmentStart = (StartTimeSince12 div 3) * 8).
```

FIG. 17e

```
CheckTimes(LocalStartTime, LocalEndTime, [FirstEvent | MoreEvents], DefiniteEvents)
:-
        // an event corresponds to the contents of the first loop from an EIT.
        is_field(FirstEvent, start_time, Start), Start >= LocalStartTime,
        is_field(FirstEvent, duration, duration), LocalEndTime >= LocalStartTime +
duration,
        Append(FirstEvent, DefiniteEvents),
        CheckTimes(LocalStartTime, LocalEndTime, [MoreEvents], DefiniteEvents).
CheckTimes(LocalStartTime, LocalEndTime, [FirstEvent | MoreEvents], DefiniteEvents)
:-
        // an event corresponds to the contents of the first loop from an EIT.
        is_field(FirstEvent, start_time, Start), Start < LocalStartTime,
        CheckTimes(LocalStartTime, LocalEndTime, [MoreEvents], DefiniteEvents).
CheckTimes(LocalStartTime, LocalEndTime, [FirstEvent | MoreEvents], DefiniteEvents)
:-
        // an event corresponds to the contents of the first loop from an EIT.
        is_field(FirstEvent, start_time, Start), Start >= LocalStartTime,
        is_field(FirstEvent, duration, duration), LocalEndTime < LocalStartTime +
duration,
        CheckTimes(LocalStartTime, LocalEndTime, [MoreEvents], DefiniteEvents).
```

FIG. 17f

```
?- event(X), eventTitle(X, Y), eventType (X, "news"), eventsTS(X, A),
TransportStreamInBouquet(A, B), ActualTransportStreamID(Z),
TransportStreamInBouquet(Z, B),
GetAllEITEvents(A, [[June, 13, 2000, 0930],[June,13,2000,1300], DefiniteEvents],
is_member(X, DefiniteEvents), networkType(A, "cable").
```

FIG. 18

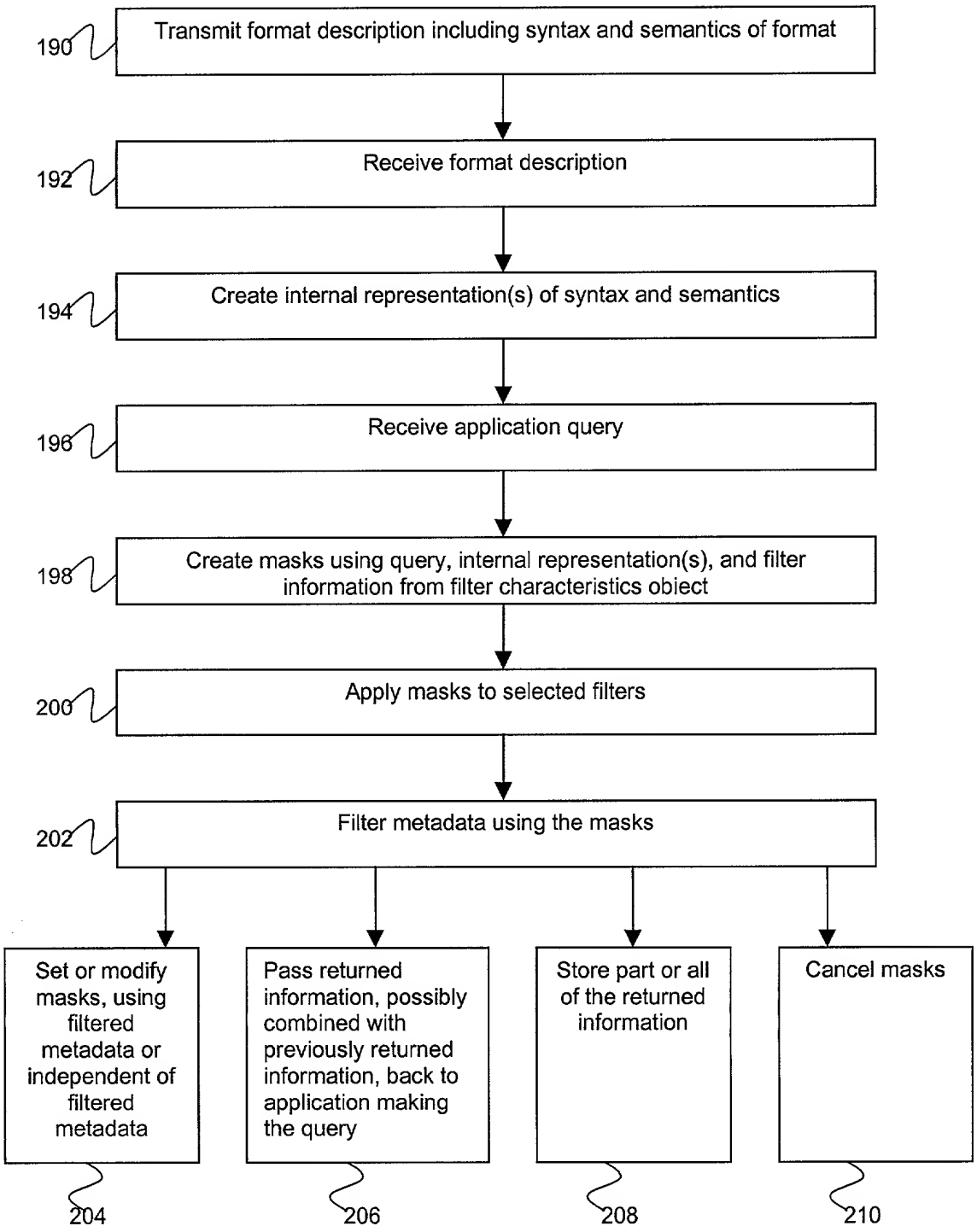
190
Transmit format description including syntax and semantics of format
192
Receive format description
194
Create internal representation(s) of syntax and semantics
196
Receive application query
198
Create masks using query, internal representation(s), and filter information from filter characteristics object
200
Apply masks to selected filters
202
Filter metadata using the masks
Set or modify masks, using filtered metadata or independent of filtered metadata
Pass returned information, possibly combined with previously returned information, back to application making the query
Store part or all of the returned information
Cancel masks
204
206
208
210

FIG. 19